ICIMP 2017

The Twelfth International Conference on Internet Monitoring and Protection

ISBN: 978-1-61208-564-7

June 25 - 29, 2017

Venice, Italy

ICIMP 2017 Editors

Eugen Borcoci, University "Politehnica" of Bucharest (UPB), Romania
Bernhard Tellenbach, Zurich University of Applied Sciences, Switzerland

ICIMP 2017

Forward

The Twelfth International Conference on Internet Monitoring and Protection (ICIMP 2017), held between June 25-29, 2017 in Venice, Italy, continued a series of special events targeting security, performance, vulnerabilities in Internet, as well as disaster prevention and recovery. Dedicated events focused on measurement, monitoring and lessons learnt in protecting the user.

The design, implementation and deployment of large distributed systems are subject to conflicting or missing requirements leading to visible and/or hidden vulnerabilities. Vulnerability specification patterns and vulnerability assessment tools are used for discovering, predicting and/or bypassing known vulnerabilities.

Vulnerability self-assessment software tools have been developed to capture and report critical vulnerabilities. Some of vulnerabilities are fixed via patches, other are simply reported, while others are self-fixed by the system itself. Despite the advances in the last years, protocol vulnerabilities, domain-specific vulnerabilities and detection of critical vulnerabilities rely on the art and experience of the operators; sometimes this is fruit of hazard discovery and difficult to be reproduced and repaired.

System diagnosis represent a series of pre-deployment or post-deployment activities to identify feature interactions, service interactions, behavior that is not captured by the specifications, or abnormal behavior with respect to system specification. As systems grow in complexity, the need for reliable testing and diagnosis grows accordingly. The design of complex systems has been facilitated by CAD/CAE tools. Unfortunately, test engineering tools have not kept pace with design tools, and test engineers are having difficulty developing reliable procedures to satisfy the test requirements of modern systems. Therefore, rather than maintaining a single candidate system diagnosis, or a small set of possible diagnoses, anticipative and proactive mechanisms have been developed and experimented. In dealing with system diagnosis data overload is a generic and tremendously difficult problem that has only grown. Cognitive system diagnosis methods have been proposed to cope with volume and complexity.

Attacks against private and public networks have had a significant spreading in the last years. With simple or sophisticated behavior, the attacks tend to damage user confidence, cause huge privacy violations and enormous economic losses.

The CYBER-FRAUD track focuses on specific aspects related to attacks and counterattacks, public information, privacy and safety on cyber-attacks information. It also targets secure mechanisms to record, retrieve, share, interpret, prevent and post-analyze of cyber-crime attacks.

Current practice for engineering carrier grade IP networks suggests n-redundancy schema. From the operational perspective, complications are involved with multiple n-box PoP. It is not guaranteed that this n-redundancy provides the desired 99.999% uptime. Two complementary solutions promote (i) high availability, which enables network-wide protection by providing fast

recovery from faults that may occur in any part of the network, and (ii) non-stop routing. Theory on robustness stays behind the attempts for improving system reliability with regard to emergency services and containing the damage through disaster prevention, diagnosis and recovery.

Highly reliable emergency communications are required by public safety and disaster relief agencies to perform recovery operations or associated with disasters or serious network events. Future advanced network development and evolution should take into consideration these requirements through solutions: (a) Identification of suitable technologies, i.e., narrowband and broadband aspects, (b) Interoperability and interworking between emergency communications capabilities and public networks, (c) Preferential access to communications resources capabilities, applications, and facilities, (d) Preferential use of remaining operational resources.

The conference had the following tracks:

- Monitoring with Web technologies
- Internet traffic surveillance and interception

We take here the opportunity to warmly thank all the members of the ICIMP 2017 technical program committee, as well as all the reviewers. The creation of such a high quality conference program would not have been possible without their involvement. We also kindly thank all the authors that dedicated much of their time and effort to contribute to ICIMP 2017. We truly believe that, thanks to all these efforts, the final conference program consisted of top quality contributions.

We also gratefully thank the members of the ICIMP 2017 organizing committee for their help in handling the logistics and for their work that made this professional meeting a success.

We hope that ICIMP 2017 was a successful international forum for the exchange of ideas and results between academia and industry and to promote further progress in the field of Internet monitoring and protection. We also hope that Venice, Italy provided a pleasant environment during the conference and everyone saved some time to enjoy the unique charm of the city.

ICIMP 2017 Chairs

ICIMP Steering Committee
Sathiamoorthy Manoharan, University of Auckland, New Zealand
Terje Jensen, Telenor, Norway
Christian Callegari, University of Pisa, Italy

ICIMP Industry/Research Advisory Committee
Daisuke Mashima, Advanced Digital Sciences Center, Singapore
Bernhard Tellenbach, Zurich University of Applied Sciences, Switzerland
Miroslav Velev, Aries Design Automation, USA
Pethuru Raj, IBM Global Cloud Center of Excellence, India

ICIMP 2017
Committee

ICIMP Steering Committee

Sathiamoorthy Manoharan, University of Auckland, New Zealand
Terje Jensen, Telenor, Norway
Christian Callegari, University of Pisa, Italy

ICIMP Industry/Research Advisory Committee

Daisuke Mashima, Advanced Digital Sciences Center, Singapore
Bernhard Tellenbach, Zurich University of Applied Sciences, Switzerland
Miroslav Velev, Aries Design Automation, USA
Pethuru Raj, IBM Global Cloud Center of Excellence, India

ICIMP 2017 Technical Program Committee

Lasse Berntzen, University College of Southeast, Norway
Abdelmadjid Bouabdallah, Université de Technologie de Compiègne, France
Aymen Boudguiga, Institute for Technological Research SystemX, France
Christian Callegari, University of Pisa, Italy
Dimitra Georgiou, University of Piraeus, Greece
Stefanos Gritzalis, University of the Aegean, Greece
Quentin Jacquemart, I3S Laboratory | University Nice Sophia Antipolis, France
Terje Jensen, Telenor, Norway
Toshihiko Kato, The University of Electro-Communications, Japan
Ayad Ali Keshlaf, Industrial Research Center, Libya
Jaime Lloret Mauri, Universidad Politecnica de Valencia, Spain
Sathiamoorthy Manoharan, University of Auckland, New Zealand
Daisuke Mashima, Advanced Digital Sciences Center, Singapore
Michael J. May, Kinneret College on the Sea of Galilee, Israel
Yisroel Mirsky, Ben-Gurion University, Israel
Constantin Paleologu, University Politehnica of Bucharest, Romania
Pethuru Raj, IBM Global Cloud Center of Excellence, India
Jani Suomalainen, VTT Technical Research Centre of Finland, Finland
Bernhard Tellenbach, Zurich University of Applied Sciences, Switzerland
Rob van der Mei, CWI and VU University Amsterdam, Netherlands
Julien Vanegue, Bloomberg L.P., USA
Miroslav Velev, Aries Design Automation, USA
Arno Wagner, Consecom AG, Zurich
Wei Wang, Nanyang Technological University (NTU), Singapore
Muhammad Azfar Yaqub, Kyungpook National University, Korea

Copyright Information

For your reference, this is the text governing the copyright release for material published by IARIA.

The copyright release is a transfer of publication rights, which allows IARIA and its partners to drive the dissemination of the published material. This allows IARIA to give articles increased visibility via distribution, inclusion in libraries, and arrangements for submission to indexes.

I, the undersigned, declare that the article is original, and that I represent the authors of this article in the copyright release matters. If this work has been done as work-for-hire, I have obtained all necessary clearances to execute a copyright release. I hereby irrevocably transfer exclusive copyright for this material to IARIA. I give IARIA permission or reproduce the work in any media format such as, but not limited to, print, digital, or electronic. I give IARIA permission to distribute the materials without restriction to any institutions or individuals. I give IARIA permission to submit the work for inclusion in article repositories as IARIA sees fit.

I, the undersigned, declare that to the best of my knowledge, the article is does not contain libelous or otherwise unlawful contents or invading the right of privacy or infringing on a proprietary right.

Following the copyright release, any circulated version of the article must bear the copyright notice and any header and footer information that IARIA applies to the published article.

IARIA grants royalty-free permission to the authors to disseminate the work, under the above provisions, for any academic, commercial, or industrial use. IARIA grants royalty-free permission to any individuals or institutions to make the article available electronically, online, or in print.

IARIA acknowledges that rights to any algorithm, process, procedure, apparatus, or articles of manufacture remain with the authors and their employers.

I, the undersigned, understand that IARIA will not be liable, in contract, tort (including, without limitation, negligence), pre-contract or other representations (other than fraudulent misrepresentations) or otherwise in connection with the publication of my work.

Exception to the above is made for work-for-hire performed while employed by the government. In that case, copyright to the material remains with the said government. The rightful owners (authors and government entity) grant unlimited and unrestricted permission to IARIA, IARIA's contractors, and IARIA's partners to further distribute the work.

Table of Contents

Development Model of a Public Safety Broadband Communications Network in Indonesia

Gerson Damanik
Development Studies Doctoral Program
Satya Wacana Christian University
Salatiga, Indonesia
gerson@postel.go.id

Denny Kusuma Hendraningrat
National Standardization Agency of Indonesia
Center for Standards Development
Jakarta, Indonesia
denny_kh@bsn.go.id

***Abstract* — The Public and Private Partnership (PPP) development model of a public safety broadband network between cellular operators and public safety agencies, such as the National Disaster Management Agency, is a challenge for the government of Indonesia to provide broadband access. Public safety agencies are local governments, police agency, health agency, fire brigades. Each agency built their networks independently. In this study, the public safety broadband network model in Indonesia is developed by using an investment budget to build a broadband network of each agency. The budget of each agency is a function of compensation for the public safety, because they do not build their own network separately, but they rather share it with the cellular network. Public safety users are included as cellular users who will be given priority access or Quality of Service (QoS), but they are not profitable users for cellular operators. So, a cellular operator only receives infrastructure compensation budget due to the addition of user traffic for public safety, because this is part of the responsibilities of the government. The feasibility of this model will be measured by Net Present Value (NPV) calculations. From a cellular operator perspective, it is concluded that operators choose the 2x25 MHz option, which must share bandwidth and network infrastructure with public safety agencies. It has a higher NPV than the 2x20 MHz option, which is only for commercial Long Term Evilution (LTE). From a government perspective, the NPV always has a positive value. So, it indicates that the government needs to consider implementing a development model of a public safety broadband network with a sharing scheme between cellular operators and public safety agencies.**

Keywords — public safety broadband network, sharing scheme, NPV, LTE

I. INTRODUCTION

Public safety is an activity comprised of prevention, treatment, and protection against things that harm other people who may be significantly affected or injured, or experience a loss or damage, such as a crime or disaster. It can be caused by human actions or a natural occurrence, which is why it is important to create a secure and comfortable condition in the community. By doing so, it can support national stability [1].

Today, communication systems supporting public safety agencies have different standards, such as using different frequency ranges of 300 MHz – 800 MHz and using different kinds of technology. The most widely used types of technology are the conventional systems, trunking systems, Public Switched Telephone Network (PSTN), and commercial cellular networks. In fact, the condition of public safety in Indonesia is still independent, which does not support interoperability among agencies. It causes coordination difficulties between agencies responding to disaster. In addition, the public safety network in Indonesia is still based on a narrowband system. The capital expenditures (capex) and operational expenditures (opex) will necessitate high investment costs when each of the public safety agencies build their own broadband networks independently. So, it will burden the government's budget while the public safety traffic is only used in emergency conditions based on operational statistic data [2] and traffic site summary information [3]. The average communication channel occupation during emergencies or disasters is 31.32 percent from the total capacity or 7.52 hours/day [4].

Consistent with the issue of broadband public safety, based on Ministerial Decree No. 22 of 2011, the Ministry of Communications and Information Technology of Indonesia has planned a migration of analog terrestrial television to digital television services, which is targeted by 2018 [5]. In Article 4 of Ministerial Decree No. 18 of 2005, it is declared that in the case where government entities desire to use a telecommunications network, they can lease it from the network provider. On the other hand, especially in Article 7 of Ministerial Decree No. 18 of 2005, it is declared that government entities networks are prohibited to collect payments [6].

Based on the explanation above, a public safety broadband network has the opportunity to integrate public safety networks, in which some portions of the Asia Pacific Telecommunity (APT) 700 MHz digital dividend bandwidth can be allocated for LTE based technology to serve public safety agencies [7]. In this study, public and private partnerships are developed to deploy a public safety broadband network in Indonesia based on the previous model [8] [9], and it has been changed according to Indonesia's condition, based on Ministerial Decree No. 22 of 2011 and Ministerial Decree No. 18 of 2005. The model is still being developed by using the existing public safety

 ISBN: 978-1-61208-564-7

network. First, it will be deployed in the greater Jakarta area and its satellite area because Jakarta, as the capital city of Indonesia, serves the central government and economy with a high population density, so it needs to have a public safety broadband system.

II. METHODOLOGY

Based on previous experiences in other countries, the Federal Communications Commission-United States (FCC-US) adopted an order to create a nationwide broadband network with a 2x10 MHz bandwidth for the Frequency Division Duplex (FDD) that consists of 758-768 MHz for an uplink and 788-798 MHz for a downlink, which is called "D Block". In America, the public safety spectrum is allocated at 763-775 MHz for an uplink and 793-805 MHz for a downlink, which consists of 2x5 MHz (763-768 MHz and 793-798 MHz) for a public safety broadband network using a bandwidth shared with an LTE network and the other spectrum allocated for a public safety narrowband network. In March of 2008, the FCC attempted to auction the D Block with public safety encumbrances but failed to attract a winning commercial bidder [10]. This is caused by several reasons, some of which include [11] [12]:

a. The 2x10 MHz bandwidth allocation in the D Block was claimed to be too small to overcome the LTE user traffic.

b. The issue has been framed in such a way as to suggest that allocations to the public safety community are at the expense of commercial wireless providers.

c. Some of the business entities collapsed and the United States (US) needs more commercial broadband network capacity to remain competitive globally.

d. The inexact time of the auction which was followed by a flurry of waiver petitions, public comments, and much debate.

Ryan Hallahan [8] improved the broadband public safety wireless communication based on the US situation, in which public safety users were reputed as being profitable users or commercial cellular customers who must pay for the use of their traffic. He devised a handover scenario whereby a handset must connect (roam) to a cellular operator if the user moves to another location which does not have public safety network coverage in Block D. In addition, APT modified 2x10 MHz of digital dividend to be allocated only for public safety communications [13].

In this research, a different method from the USA is deployed. In Indonesia, it is developed from a public partnership model between cellular operators and public safety agencies, where the public safety users are cellular users that will be given priority access or quality of service (QoS), but they are not considered as profitable users for cellular operators. Public safety user traffic on a cellular network will be converted to the additional costs (capex and opex) of cellular network deployment. In a government perspective, the investment cost payments should be managed by the government, as the Ministry of Finance should provide the budget for the public safety broadband network agencies. In this study, those payments are defined as a function of the government costs. This model developed the investment cost utilization as a budget which is canceled for each of the public safety agencies to build a public safety broadband network independently. In this study, that canceled budget is defined as a function of the government value. The government should consider the expenditure efficiency when deploying a model of a public safety broadband network, so that the feasibility will be measured from the NPV of a government perspective. From a cellular operator perspective, the government cost is a portion of the contributions to the cellular network as an operator value function of the cellular operator NPV, besides the annual revenue per user (ARPU) of commercial users. In this model, the operator costs are calculated from the total investment sharing network costs and annual spectrum fees.

III. NPV MODEL DEVELOPMENT

This model is developed from the previous studies of Ryan Hallahan [8], John Ure [9], and Administrative Incentive Pricing (AIP) recommended by Australian Communication and Media Authority (ACMA) [14] with an adoption of the conditions of Indonesia. An illustration of this model can be viewed in Figure 1.

In this study, the NPV formula is based on a government and cellular operator perspective and developed as a measure of examining the feasibility of developing a public safety broadband network based on the model proposed in this study. The cellular operator NPV during the observation is defined by t = i, as follows:

$$NPV_{Op} = \sum_{i=0}^{n} \frac{(GV_i + 12.\ Sub_{COMM,i}.R_{COMM}) - (C_i.\ Capex + C_{TOT,i}.\ Opex)}{(1+D)^i} - SFi \quad (1)$$

Then, the government NPV is formulated as follows:

$$NPV_{Gov} = GV_i - GC_i \quad (2)$$

Where,

GV_i = total investment cost utilization as a budget, which is cancelled by the year-i for each of the public safety agencies to build a public safety broadband network independently. [USD/year]

GC_i = total payment of the investment costs by the i-year which should be prepared by the government, as a compensation for the public safety user traffic to the cellular operator. [USD/year]

$Sub_{COMM,i}$ = total number of commercial subscriptions by the year-i.

RCOMM	=	monthly revenue from commercial subscriptions.	[USD/month]
Ci	=	amount of equipment per element developed in the i-year.	
CTOT,i	=	total amount of equipment per element operating in the i year.	
Capex	=	upfront cost to develop the network per element.	[USD]
Opex	=	annual cost to operate the network per element.	[USD/year]
SFi	=	Annual Spectrum Fee (LTE 700 MHz) in the i-year.	[USD/year]
n	=	time horizon.	[Years]
D	=	discount rate.	[%]

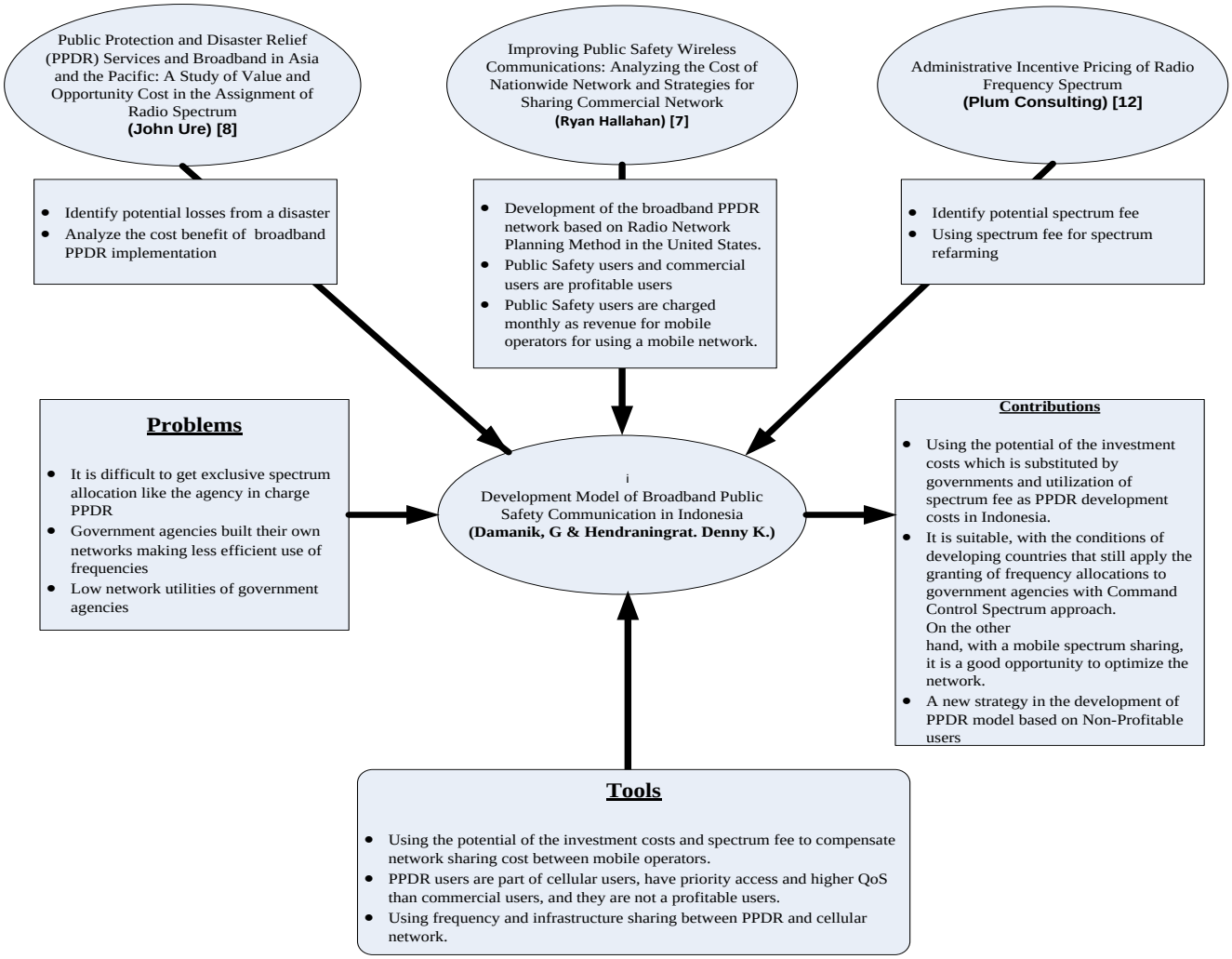


Figure 1. Development Model of a Public Safety Broadband Network

IV. Results and Feasibility Analysis of Public Safety Broadband Network Communications in Indonesia

The 3rd Generation Partnership Project (3GPP) has identified a 2x45 MHz bandwidth allocation for the Asia Pacific Region as a bandwidth allocation for Evolved Universal Terrestrial Radio Access (E-UTRA) technology, such as LTE technology [15]. In this study, the digital dividend ecosystem is divided into commercial LTE and public safety. LTE is more effective than Dual Carrier of High Speed Packet Access (DC-HSPA) when using a 2x20 MHz bandwidth system [16]. This simulation is designed by using 2 Mobile Network Operators (MNOs), where the MNO that is willing to share bandwidth and network infrastructure with the public safety agencies will be given a

2x25 MHz bandwidth allocation and the other MNO will be allocated 2x20 MHz.

In this model, public safety users are cellular users who will be given priority access. The standard broadband QoS is described by 2 Mbps user throughput [17]. The services provided to the public safety broadband network include voice, two-way video, and data transfer.

In this study, the feasibility of broadband public safety communication is measured based on the NPV calculation, both from the government and cellular operator perspectives [18]. It consists of calculating the network (coverage and capacity) planning and then calculating the network cost deployment, so that the NPV can be determined.

A. *Defining Network Planning for Public Safety Communication*

1) Coverage Planning: This computation focused on performing a calculation of a maximum cell range of LTE 700 by QoS, which is outlined in Table I. In this study, it is assumed that the use of broadband LTE is in a fixed outdoor area. Based on the coverage planning method, the propagation conditions are one of the main factors to determine the cell size. In this study, link budget simulations are conducted to know the number of LTE e-Nodes B, which are needed to cover the planning area. The cell range prediction is calculated by adopting Okumura Hatta's [19] propagation model. An example of an LTE link budget calculation is shown in Table I.

TABLE I. LTE COVERAGE PLANNING

Main Parameters	Dense Urban	Urban	Suburban	Rural
Freq Operation (MHz)	700			
RF PA power (Watt)	20			
Channel BW (MHz)	20			
Cell Edge Tput DL (Kbps)	2,048			
RF Load	80%			
BTS Antenna Height (m)	30	30	40	70
Cell Range (km)	1.62	1.95	4.89	6.80
Cell Area (km2)	1.71	2.47	15.55	30.05
Site Area (km2)	5.13	7.40	46.64	90.16
Inter Site Distance (km)	2.43	2.92	7.34	10.20

2) Capacity Planning: In a cellular network, capacity planning is required for the network optimization to meet the QoS requirements [20]. The calculation of capacity planning is started with an LTE rollout plan and the user prediction of the Indonesia cellular provider which has a 43% market share. So, the number of eNodes-B is calculated using the following formula [21]:

$$Number\ of\ eNodeB = \frac{Total\ Throughput\ Offered}{Site\ Throughput} \quad (3)$$

3) Defining Network Cost Deployment: Based on data from the vendor, the network infrastructure costs were calculated for the components, as shown in Table II.[22]

TABLE II. INFRASTRUCTURE COSTS

CAPEX	Price (USD)	Notes
RAN	71,190	per unit
Core	12,000,000	per unit
Data Center and Application	714,695	per unit
OPEX		
RAN	35,171	per unit
Datacomm and Transport	16,268	per unit
Core	21,367	per unit
Data Center and Application	545	per unit

The total investment costs were calculated by multiplying the results of the network planning with the price list, which is shown in Table II. The total investment costs required to build the LTE network with a sharing system (first option) between a cellular operator and public safety is shown in Figure 2.

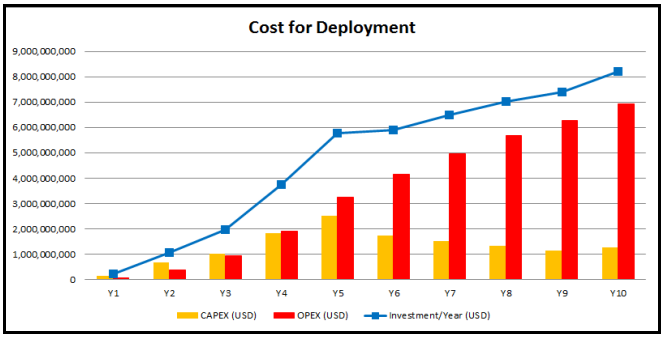


Figure 2. Total Investment Costs

B. *Defining NPV Calculation*

The NPV calculation is developed on the basis of revenues minus total expenses. From a government perspective, the revenue or government value is the total investment cost utilization as a budget which is cancelled by public safety agencies to build a public safety broadband network independently. On the other hand, the government cost is the total payment of the investment costs by the year-i which should be prepared by the government, as a compensation for public safety user traffic to the cellular operator. Figure 3 shows the calculation of government value and government costs.

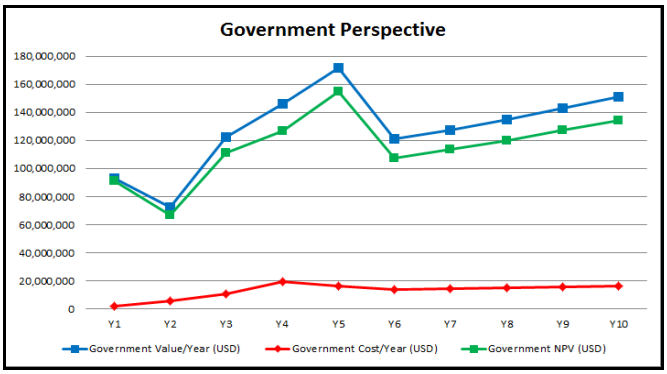


Figure 3. Government Perspective

 ISBN: 978-1-61208-564-7

From the cellular operator perspective, the operator revenue (operator value) is the annual revenue per user (ARPU) of commercial users plus the compensation costs from the government (government costs). On the other hand, the operator costs are calculated from the total investment sharing network costs and annual spectrum fees. In the first year, the government value has a high value obtained from the capex (core networks) of public safety agencies to build a public safety broadband network independently. In the second year, the public safety network is not required to build core networks (only towers and e-NodeB). In the sixth year, the public safety network only requires maintenance fees (opex). So, if the Ministry of Finance diverts the costs of public safety agencies to build a public safety broadband network independently to become a sharing model, then it will be advantageous for the government. Figure 4 shows the calculation of operator value and operator cost.

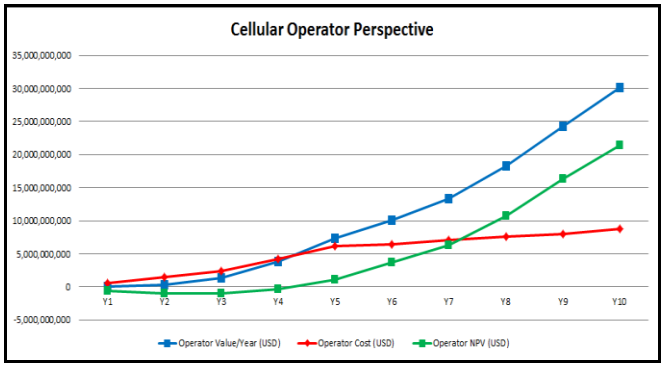


Fig 4. Cellular Operator Perspective

Figure 4 shows that the operator NPV has a positive value after the 5th year of LTE deployment. A cellular operator's revenue always increases after the 5th year of LTE deployment. It is concluded that cellular operators need to consider implementing the LTE technology.

C. Simulation Results of NPV Calculations

1) Cellular Operator Perspective: In this scenario, the cellular operator is only given two options of bandwidth allocation. This simulation will compare the results of the NPV calculation between these two options. In the first option, the operator using 2x25 MHz must share the bandwidth and network infrastructure with the public safety agencies. Based on the APT recommendation [16], public safety agencies will be given 2x10 MHz dedicated only for public safety communication. However, in this development model, it is designed with 2x10 MHz for sharing between public safety and commercial LTE and 2x15 MHz only for commercial LTE. In other words, the maximum bandwidth allocation is 2x25 MHz for commercial LTE and 2x10 MHz for public safety communication. In the second option, the operator only uses 2x20 MHz for commercial LTE. Figure 5 shows the NPV results for the first option (2x25 MHz) and second option (2x20 MHz) while setting a discount rate at 5% [18].

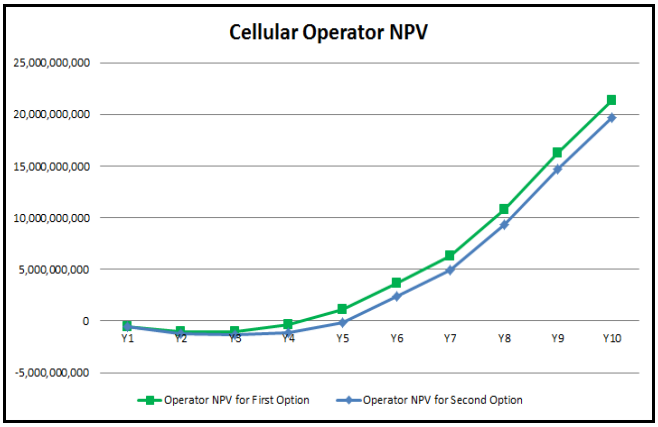


Figure 5. Cellular Operator NPV

Figure 5 shows that the NPV results for first option are higher than the second option. It is concluded that a cellular operator will obtain more benefits if the first option is taken rather than the second option.

2) Government Perspective: In this model, the Ministry of Communications and Information Technology acts as a grantor of the sharing policy between cellular operators and public safety agencies. On the other hand, the Finance Ministry acts as the owner of the budget for financing public safety broadband network implementation with a sharing concept between cellular operators and public safety agencies. Figure 6 shows the NPV results based on the government's perspective.

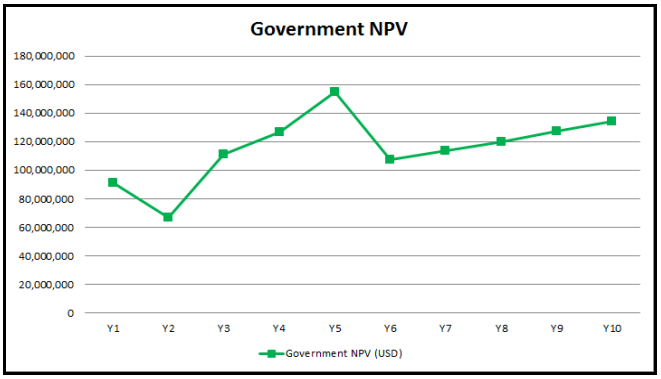


Figure 6. Government NPV

Figure 6 shows that the NPV results have a positive value with the implementation of this sharing model. This suggests that the government needs to consider implementing the development model of broadband public safety through a sharing scheme between cellular operators and public safety agencies.

 ISBN: 978-1-61208-564-7

V. CONCLUSION

In this study, a public and private partnership model is developed to deploy the public safety network through a sharing model with commercial cellular operators. In this model, the investment cost utilization is a budget which is canceled for each of the public safety agencies to build a public safety broadband network independently. This study contributes to the cost savings of public safety network development.

The feasibility of this model is measured by net present value (NPV) calculations. From the cellular operator perspective, it is concluded that operators prefer the 2x25 MHz option, which must share bandwidth and network infrastructure with the public safety agencies. It has higher NPV than the 2x20 MHz option only for commercial LTE. From a government perspective, the NPV always has a positive value. So, it indicates that the government needs to consider implementing Development Model of a Public Safety Broadband Communications Network through a sharing scheme between cellular operators and public safety agencies.

REFERENCES

[1] G. Eason, B. Noble, and I. N. Sneddon, “On Certain Integrals of Lipschitz-Hankel Type Involving Products of Bessel Functions,” Phil. Trans. Roy. Soc. London, vol. A247, pp. 529–551, April 1955.

[2] G. Damanik, “Determining the Number of e-Node B for Digital Dividend Public Safety Communication in Jakarta Area,” ECTI International Journal. Thailand.

[3] Jakarta Local Government., “Operational Statistic of Trunking Analog,” Jakarta, Indonesia. 2013.

[4] Polda Metro., “Traffic Site Summary of Land Mobile Radio (LMR)”. Jakarta, Indonesia.

[5] Regulation of the Minister of Communications and Information Technology of Indonesia No. 22 of 2011.

[6] Regulation of the Minister of Communications and Information Technology of Indonesia No. 18 of 2005.

[7] ITU-R M. 2015 Frequency Arrangements for Public Protection and Disaster Relief Radiocommunication Systems in UHF Bands in Accordance with Resolution 646 (Rev.WRC-12).

[8] R. Hallahan, “Improving Public Safety Wireless Communications: Analyzing the Cost of Nationwide Network and Strategies for Sharing Commercial Network,” Carnegie Mellon University. 2011.

[9] J. Ure, “Public Protection and Disaster Relief (PPDR) Services and Broadband in Asia and the Pacific: A Study of Value and Opportunity Cost in the Assignment of Radio Spectrum,” TPRC Corporate. 2013.

[10] M. L. Goldstein, “Emergency Communication-Various Challenges Likely to Slow Implementation of a Public safety Broadband Network,” United Stated Government Accountability Office.

[11] Hatfield and Dawson, “Discussion of 700 MHz Spectrum Policy Issues for Public Safety in King Country,” NetCity Inc. 2010.

[12] T. Takai and M. Bettenhausen, “California Public Safety Radio Communications Strategic Plan,” The Great Seal of The State of California. 2010.

[13] T. Welter, “Assessing the Potential of The 700 MHz Band For PPDR,” Amsterdam. 2014.

[14] Plum Consulting., “Administrative Incentive Pricing of Radio Frequency Spectrum,” London. 2008.

[15] 3GPP TS 36.101 V9.4.0, “Technical Spesification Group Radio Access Network; Evolved Universal Terrestrial Radio Access (E-UTRA); User Equipment (UE) Radio Transmission and Reception.”

[16] H. Holma and A. Toskala, “LTE for UMTS-OFDMA and SC-FDMA based Radio Access,” John Wiley & Sons Ltd. 2009.

[17] D. Setiawan, D. Sirat, and D. Gunawan, “Feasibility of LTE 700 MHz Digital Dividend for Broadband Development Acceleration in Rural Areas,” ITB Journal of Information and Communication Technology. 2012.

[18] D. Setiawan, D. Sirat, and D. Gunawan, “Acceleration Model of Digital Dividend Implementation in Indonesia,” University of Indonesia. 2013.

[19] D. K. Hendranigrat, N. F. Adriansyah,. U. K. Usman and D. Setiawan, “Refarming Analysis of 700 MHz Frequency Band for Long Term evolution (LTE) Implementation In Indonesia Using Link Budget Calculation,” International Conference on Electrical Engineering and Informatics (ICEEI). Bandung, Indonesia; ISSN : 2155-6822. ISBN : 978-1-4577-0753-7. volume 3: page.1688-1692, 2011.

[20] D. K. Hendraningrat, “Analysis of Slot Spectrum Selection For Long Term Evolution (LTE),” The 5th International Conference TSSA. Bandung, Indonesia. ISBN : 978-1-4577-1441-2. 2009.

[21] S. A. Basit, “Dimensioning of LTE Network Descripttion of Model and Tool, Coverage and Capacity Estimation of 3GPP Long Term Evolution Radio Interface,” Helsinki University of Technology. 2009.

[22] D. Setiawan, and D. K. Hendraningrat, “Digital Dividend Implementation Acceleration in Indonesia,” The 8th International Conference TSSA. Bali, Indonesia. ISBN : 978-1-4799-4774-4. 2014.

 ISBN: 978-1-61208-564-7

Honey-Copy - A Concept and Prototype of a Generic Honeypot System

Olivier Favre*, Bernhard Tellenbach*, Jan Alsenz†
*Zurich University of Applied Sciences, Switzerland
†Oneconsult AG, Switzerland
email: {favr,tebe}@zhaw.ch*, jan.alsenz@oneconsult.com†

***Abstract*—In this paper, we present Honey-Copy, a concept and prototype for a honeypot system that can pinpoint modifications caused by attacks or intrusion for any honeypot. To achieve this, we track modifications without having to install any additional tools on them. We make use of cloning to identify whether or not a modification has been caused by the honeypot itself or an attacker or intruder. We briefly present our initial prototype and discuss the challenges to be solved toward a more complete and feature rich version of our prototype.**

Keywords–Honeypot; Detection; Security; Monitoring;

I. INTRODUCTION

Honeypots are decoy computer resources whose value lies in being probed, attacked or compromised [1]. The main difference between a normal computer resource and a honeypot is that the honeypot is not part of the production infrastructure [2]. One notable exception is the concept of Shadow Honeypots presented in [3]. As a consequence, attack detection methods do not have to cope with arbitrary production activity and the extraction of traces of attacks or intrusions is much simpler. After all, the traces do not submerge in production activities [2]. Honeypots are therefore a valuable tool to improve detection and reaction. However, since they do not protect a production infrastructure directly, they must be integrated with traditional security controls [4].

The lack of off-the-shelve products and solutions that allow automated and easy creation and monitoring of honeypots might be one of the reasons why the list of security controls used by a company does rarely contain one. Another reason might be that even though there exists many different kinds of honeypot systems and methodologies to analyze data produced by them, there is no system that satisfies all of the following four properties [2]: (1) the honeypots are not recognizable as such, (2) they are easy to configure and deploy, (3) the system reports activities related to attacks and intrusions only, (4) the core mechanisms (deployment, reporting of activities) work for any honeypot. Properties one and three are probably the most important ones. If these are not met, the system is of limited use since it would be easy to detect and it would be difficult to extract useful information from its reports. Properties two and four are relevant from an operational and business perspective. One of the major challenges is finding a solution to the problem of reporting activities related to attacks and intrusions without having to craft honeypot-specific algorithms or rules. At first glance, assuming that any activity on the honeypot is suspicious and should therefore be reported seems like a simple solution to this problem. After all, there is no production activity on a honeypot. While this often-made assumption might hold for activities like incoming network connections, it does not fit activities like the creation of a process or the modification of a file. Depending on the honeypot itself, we might see a significant amount of activity even on an ”idle” system. This includes things like automated software updates, an application-specific timed or event-based tasks (e.g., sync or cleanup tasks) or log entries from arbitrary scheduled tasks. Furthermore, when considering property four, the assumption about incoming network connections might be wrong too - a honeypot might do updates using active FTP or it might run a distributed service that sees incoming connections from other parts of the service from time to time. It is therefore crucial to have a generic way to distinguish between activities of type *self* and *third party* with the former including any activity triggered (or expected) by the honeypot itself.

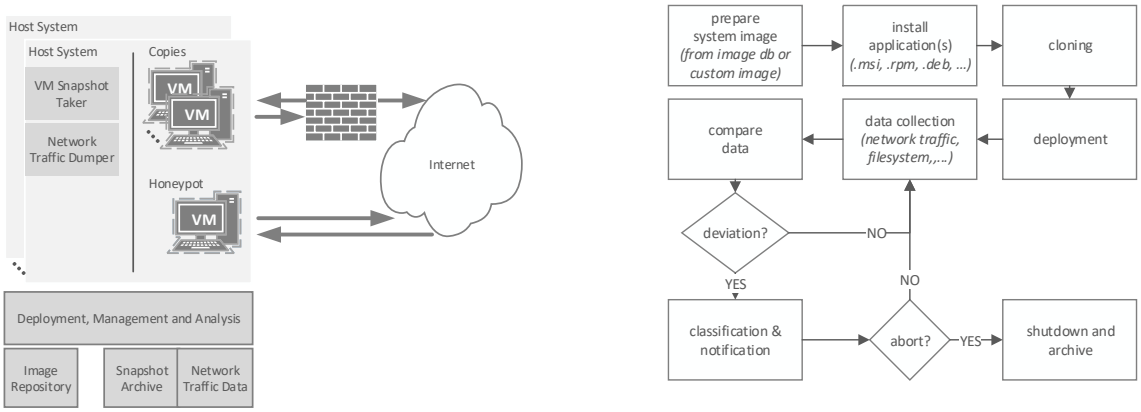
In this paper, we present the main idea and concept of Honey-Copy, a system that should overcome most of the limitations of today’s honeypot systems. Our main contribution is a generic method to distinguish between activities of type *self* and *third party* and its integration in a general concept for a honeypot system. First, we provide an overview of Honey-Copy and discuss the basic idea of our generic approach to identify whether or not an activity is triggered by the honeypot itself or a third party (Section II) and we explain why an implementation of such a system is likely to be limited to high-interaction honeypots. Next, we introduce our prototype and discuss its implementation and evaluation (Section III). We then conclude our paper with a section on the challenges and next steps toward a more advanced version of our prototype (Section IV). A discussion of relevant related work can be found at the end of the paper (Section V).

II. HONEY-COPY - BASIC CONCEPT

The core idea of Honey-Copy is to make a clone of a honeypot and to put it behind a firewall that blocks incoming network connections. Since the clone cannot be accessed by third-parties, it exhibits activities of type *self* only. It should therefore be possible to identify and filter those activities from the reports of the honeypot system exposed to the attackers. Unfortunately, things like applications that create (temporary) files with random filenames or software updates that happen at different points in time make it difficult to implement an accurate and timely matching. This is why Honey-Copy can make up to n clones of a honeypot. By comparing them, patterns like random filenames can be identified and the chance that honeypot and clone(s) exhibit a certain activity of type *self* within a short time span can be increased. Hence, it should be possible to satisfy property (3).

Figure 1a shows the basic building blocks and setup of Honey-Copy. It consists of physical machines that make use of virtualization to run a host system and potentially many guest systems. The use of virtualization enables Honey-Copy to clone and deploy anything that can be provided in the form of a virtual machine image. This meets property (4) with respect to deployment. But this is not the only benefit of

 ISBN: 978-1-61208-564-7



(a) (b)

Figure 1. (a) Shows the basic setup of the Honey-Copy system and (b) the basic procedure it uses to deploy and manage a honeypot.

virtualization. Its use enables us to do the tracking of activities in the guest systems without installing any additional software and change to their configuration. In contrast to other honeypot systems, the honeypots deployed can be virtually identical to the production system they pose as. Hence, Honey-Copy can be said to satisfy property (1); the honeypot system itself does not make its honeypots more recognizable as such by itself.

The other two components running on the host system are the *VM Snapshot Taker* and the *Network Traffic Dumper*. The former can take snapshots of the guest systems, for example file-system or memory snapshots. The later can dump information about their network activity, for example full packet traces or flow level information only. This data can be captured and added to the *Snapshot Archive* or the *Network Traffic Data Archive*. It is then analyzed by the *Deployment, Management and Analysis* component, the heart of Honey-Copy. With the data available about the systems, different kinds of information like file or registry changes, running processes or network traffic can be checked for changes. From a data analysis point of view, the problem of identifying activities of type *self* by comparing this data from the firewalled clones and the honeypot itself is largely independent of the actual honeypot. The component has to learn or identify activities of type *self* from the clones and filter them from those reported by the honeypot. Hence, since the deployment mechanism as well as the activity-reporting works for any honeypot, Honey-Copy can be said to satisfy property (4).

In addition to the analysis task, it is also responsible for managing an Image Repository and for customizing and deploying images upon request. Unfortunately, it seems difficult to implement this in a generic and easy-to-use way for arbitrary honeypot types and configurations. One option to satisfy property (2) is to implement Honey-Copy for high-interaction honeypots only. In contrast to low- and medium interaction honeypots, these are real systems and not (partially) emulated or simulated ones. It seems, for example, practical to create a repository of images for many different operating systems and to use their software packaging and configuration mechanisms to quickly make and deploy a system that is a copy of a production web server or any other server or computer in a company network. In a corporate environment it would also be possible to use production server templates and mechanisms as a basis for this process.

Figure 1b illustrates the basic procedure to setup and manage an arbitrary high-interaction server honeypot in the Honey-Copy system. The first step is to choose an image from the *Image Repository* or to provide a custom image via an upload function. Next, additional applications and services can be installed and configured by providing them as package in the packaging format of the operating system, for example .msi for Windows or .deb for Debian Linux images or by using the configuration or packaging tool used by the organization. The packaging must allow for a fully automated installation of the application or service. Now that the image is ready, it can be cloned *n*-times and the honeypot and its clones can be deployed on the same or multiple physical hosts as outlined in Figure 1a. Data collection starts at the same time as the honeypot and its clones are turned on and does not stop until this honeypot is shutdown. In regular intervals, the data collected by the clones and the honeypot is compared and if deviations are found, they are classified and notifications are sent to those that subscribed to them. It is then checked whether the deviation found requires taking the honeypot offline, for example because it was hacked and the intruder started to attack 3rd parties in the Internet.

III. PROTOTYPE

In theory, the Honey-Copy concept meets all of the four desired properties. However, it is unclear whether or not it can be put into practice. To understand the related problems and challenges better, we built a an initial prototype of Honey-Copy.

A. Implementation

Our prototype consists of some Python scripts and a set of tools orchestrated by them. To manage and deploy the honeypots and their clones, we make use of Vagrant [5], a tool that is often used to create and configure lightweight, reproducible, and portable development environments. To deploy a honeypot, we first create a configuration file that specifies the type of machine to be used, the software that needs to be installed, and

 ISBN: 978-1-61208-564-7

the way to access it. Based on this file, Vagrant can then create, deploy and launch an image for VirtualBox [6], a hypervisor that integrates well with Vagrant. When doing so, our prototype makes sure that the honeypot is cloned and that data capturing and the processes to detect activities other than *self* are in place and started.

For now, data capturing consists of recording full packet traces with tcpdump and snapshots of the file systems every *T=3600* seconds. This interval of one hour was chosen mainly to investigate the longer-term deviations between the clones and provide examples for activities of type *self*. For an actual detection setup, a much smaller interval is expected to be put in place. Whenever a new set of snapshots has been taken, the file systems of the honeypot and the clones are reconstructed, mounted and then scanned for differences using rsync. The reconstruction is required because we take differential snapshots to save storage space. In parallel to the file-system analysis, Pyshark with custom filters and rules (IP-Addresses, DNS-Names) is used to scan the network traffic dumps to extract communication partners that have not been seen by the clones. The result of the detection process is a report consisting of the differences in the file systems and the communication parters that are unique to the honeypot.

B. Evaluation

For an initial evaluation of our prototype, we deployed and tested the system with Linux and Windows based clones of typical web servers. For the evaluation, we compare the current status of the system to a perfect implementation of the Honey-Copy concept in terms of stealthiness, ease of deployment, attack detection and generic core mechanisms:

Stealthiness: The only two limitations of the prototype are that taking a file system snapshot of a virtual machine requires to suspended it and that the honeypots and clones are not physical but virtual machines. An attacker could detect the former using well-timed queries to the machine and the later might be achieved using fingerprinting methods like [7]. But the virtualization solution also supports snapshots of running machines, which could be implemented to mitigate the first limitation and as most organizations are using virtual machines at least in parts of their production infrastructure, this fact cannot be used as a sole indicator for a honeypot. Additionally, the only trace of Honey-Copy in the guest system is Puppet, an open-source software configuration management tool for Windows and Unix-like systems which is installed on them by Vagrants provisioning system. However, unlike honeypot specific logging and monitoring tools, its presence is not telling very much and it can be easily replaced with other tools. Other limitations exist but they are not introduced by the prototype itself but depend on how the system and its environment is configured and operated. For example, a public hostname like honeypot.company.com could be suspicious when used for a web server. And a system running a discussion forum with no activity in it might also look suspicious. As these factors are outside of the control of our solution and can be highly application specific, we consider them as out of scope for the prototype.

Ease of deployment: The prototype comes with the basic mechanisms and capabilities required to implement a user-friendly and easy-to-use interface to configure and deploy honeypots. However, for now it, the only interface is a command line interface. Furthermore, the *Image Repository* contains a few base images only.

Attack detection: The current mechanisms used for filtering activities of type *self* produces a significant number of false positives. One reason for this is that the prototype compares the file systems of the honeypot and its clones using the most recent snapshot only. For example, we observed many false positives because of automated software updates that did not happen or finish within the same snapshot interval. Another reason is that the comparison uses exact file matching. This turns files that are semantically the same but that have a different filename (e.g., temporary files with random filenames) or content (e.g., logfiles) into false positives. Another limitation is linked to the report generated from comparing the network traffic to the honeypot and the clones. This report lists communication partners seen by the honeypot but not its clones. Unfortunately, it contains a lot of entries that are not really interesting. This includes for example legitimate partners like search engine bots or Shodan [8] or illegitimate ones doing reconnaissance using known methods and tools. Furthermore, because the comparison of communication partners is done using exact matching, it cannot cope well with endpoints like content distribution networks.

Generic core mechanisms: Management and deployment works with any honeypot that is based on Windows or a Unix-like systems since these are the systems supported by Vagrant/VirtualBox. The same is true for the data capturing and comparison mechanisms since it does not depend on the actual system run in VirtualBox. Note that Unix-like includes most Linux distributions, Android and Mac OS.

IV. Challenges and Next Steps

In summary, we can identify two main challenges that the next version of our Honey-Copy must address. First, the system must provide a user-friendly and easy-to-use interface to configure and deploy honeypots. This can be done by writing an graphical user interface that compiles settings like the base images and the applications to be used by the honeypot into a suitable Vagrant file. The second challenge is more difficult to address. The mechanisms to identify files that are not modified by activities of the honeypot itself have to be able to detect files that are identical from a semantic point of view but that differ in content and/or have a different file name. To achieve this, generic heuristics that can detect patterns in file names or in the content of the files could be used. Another option would be to employ machine learning to search for such patterns. Furthermore, to cope for changes that might happen at different points in time on the honeypot and the clones, the mechanism must consider multiple snapshots from different points in time. What this means in terms of a delayed reporting and alerting is an important point of the evaluation of such an approach. While the focus is clearly on the file system part, there is also room for improvement with respect to the communication partners (attackers) reported by Honey-Copy. Endpoints like Windows update servers should not be reported as problematic because the honeypot and the clones use a different server for the update. The main challenge here is to make the matching mechanism aware of content distribution networks and similar behavior, for example by using third party tools, domain name resolution analysis or URL based heuristics to detect them.

 ISBN: 978-1-61208-564-7

We plan to address these challenges in the next version of our prototype.

When these have been addressed, there are still many more ways that the Honey-Copy prototype could be improved. If we consider that activities of category *third party* can be subdivided further into *benign*, *attack* and *intrusion*, it becomes clear that depending on the purpose of the honeypot, it could be required that Honey-Copy can filter activities of type *benign* and maybe even *attack*. *benign* activities are triggered accidentally or without malicious intent. This includes scanning from legitimate sources like Shodan HQ [8] or search engine bots, connection attempts that are the result of someone mistyping an IP address or URL and backscatter [9] traffic. Activities of type *attack* are triggered by an attempt to compromise the honeypot, for example using the Metasploit framework [10] and those of type *intrusion* are triggered by a successful compromise of the honeypot. To identify them, it could be useful to correlate network and file system activities and to employ an intrusion detection systems like Snort or Bro to fingerprint known attacks. We plan to research whether and how this could be done without having to sacrifice the generic nature of Honey-Copy when moving toward the third version of our prototype. Any other improvements like for example the addition of memory snapshots to the sources of information, is left to prototypes beyond version three.

V. Related Work

High-interaction honeypot systems that have similar goals in terms of stealthiness, attack detection, ease of deployment and honeypot configurations (operating system, applications etc.) are HI-HAT [11], HoneyBow [12], and Sebek [13]. Like Honey-Copy, these systems are server honeypot systems. In addition, we review a number of projects in the client honeypot sphere that are interesting because of the way they approach the problem of differentiating between real attacks and other activities.

HI-HAT [11] implements a system which converts normal PHP web applications into usable server honeypots. Their solution mainly consist of two components: The first component converts an arbitrary PHP application into a honeypot by adding monitoring capabilities to functions that handle requests from the outside. The second component consists of a GUI which lets an operator analyze the data gathered by the honeypot. To decrease the amount of false positives (generated by web crawler or other legitimate requests) the system makes use of white- and blacklisting based on general attack patterns. Furthermore, it allows the creation of custom filters to take into account different behavior of applications. Similar to Honey-Copy, it implements a way to deal with false positives generated by generic PHP applications.

HoneyBow [12] on the other hand is a high-interaction server honeypot which is designed for generic applications. It makes use of virtual honeypots to automate the management and monitoring of the system. In order to collect the necessary data to detect an attack, it implements three different tools (MwWatcher, MwFetcher, MwHunter) that search for malware binaries in the virtual filesystem and the network flow. Similar to the methods used in Honey-Copy, the MwFetcher component compares the content of the honeypot filesystem to the one of a clean copy that was taken at the start of an operation. The files which were new or altered and are flagged as executables are then further processed as malware. MwWatcher on the other hand is installed on the honeypot itself and can detect changes to the filesystem in real time. MwHunter finally inspects the network traffic for packages that contain executable malware. Each of the tools used in Honeybow has its own advantages and limitations. The MwWatcher component for example can be easily detected and disabled by an attacker. While this approach increases the chance of detecting an attack it also decreases the number of attacks since the system can be easily identified as a honeypot. Furthermore, it cannot deal with updates on the honeypot since that would likely change a number of executables compared to the clean copy.

Sebek [13] is another popular high interaction server honeypot system. It provides a data capture tool which monitors all actions of an intruder by capturing all `sys_read` calls. Furthermore, it tries to capture and send the logged data as stealthy as possible. Nevertheless, there have been a number of publications, notably [14], which show a way to detect and even disable Sebek. Another limitation of the tool is that there is no filter for the captured data. A manual analysis is required to distinguish a real attack from a false positive caused by normal system activity.

In the area of client honeypots Capture-HPC [15] and Capture Bat [16] present similar ideas. Both systems are high-interaction honeypots that make use of exclusion and inclusion lists. The former specifies acceptable non-suspicious activities to be ignored by the detection mechanisms. The later contains activities that are considered to be malicious. Such lists can be created for resources like the Windows registry, the file system, or processes. Capture-HPC also supports regular expressions to group a number of exemptions together. Currently, these list have to be created by hand and both systems run on Windows only. Another limitation of the approach is that any change to the honeypot (software updates, different mix of applications, etc.) is likely to require a modification of the exclusion list. UW-Spycrawler [17] on the other hand, makes use of trigger conditions (blacklists) which are specific to the browser used in the client honeypot setup. These conditions define activities which cannot be caused by the browser itself. Similar to the use of whitelists, these lists have to be created manually for a specific application (browser). Shelia [18] on the other hand takes a different approach to the problem without white- or blacklisting. The researcher behind the project proposes a system where the focus is on a reduction of false positives. It gathers data of an attack by monitoring the registry changes and file system actions generated by a process. The detection of said attack is done by analyzing from which memory address an API call was invoked. Once this address is obtained, it is checked whether it points to an executable memory location. If this is not the case an alarm is generated. This method allows to detect buffer and heap overflows that are exploited by an attacker. The downside of such a system is that it produces a higher number of false negatives since there are ways to circumvent the detection [19]. Pwnypot [20] take this idea even further by implementing more methods to detect arbitrary shellcode. It can detect ROP-Exploits and ASLR/DEP-Bypasses used by attackers.

VI. Conclusion

The main contribution of this paper is a concept and an initial prototype of Honey-Copy, a system that uses cloning

 ISBN: 978-1-61208-564-7

to address the problem of distinguishing activities of the honeypot itself from those of attackers. We explain why and how our concept could be used to build a honeypot system that comes close to a perfect one in terms of stealthiness, ease of deployment, reporting of activities triggered by attackers and independence of the core mechanisms from the actual honeypots to be deployed. Other systems violate at least one of these properties. Our evaluation of the prototype shows that the basic mechanisms of our concept work and allow for a stealthy and generic implementation of the system. However, to satisfy all of the properties, the current method to compare the state of the clones to the state of the honeypot has to be replaced by a more sophisticated one and an easy-to-use graphical interface to configure and deploy a honeypot has to be developed.

REFERENCES

[1] L. Spitzner, "The Honeynet Project: trapping the hackers," IEEE Security Privacy, vol. 1, no. 2, Mar 2003, pp. 15–23.

[2] M. Nawrocki, M. Wählisch, T. C. Schmidt, C. Keil, and J. Schönfelder, "A Survey on Honeypot Software and Data Analysis," e-print arXiv:1608.06249, Aug. 2016.

[3] K. G. Anagnostakis, S. Sidiroglou, P. Akritidis, K. Xinidis, E. Markatos, and A. D. Keromytis, "Detecting Targeted Attacks Using Shadow Honeypots," in Proceedings of the 14th Conference on USENIX Security Symposium - Volume 14, ser. SSYM'05. Berkeley, CA, USA: USENIX Association, 2005, pp. 9–9. [Online]. Available: http://dl.acm.org/citation.cfm?id=1251398.1251407

[4] E. Vasilomanolakis, S. Karuppayah, P. Kikiras, and M. Mühlhäuser, "A Honeypot-driven Cyber Incident Monitor: Lessons Learned and Steps Ahead," in Proceedings of the 8th International Conference on Security of Information and Networks, ser. SIN '15. New York, NY, USA: ACM, 2015, pp. 158–164. [Online]. Available: http://doi.acm.org/10.1145/2799979.2799999

[5] "Vagrant," HashiCorp, URL: https://www.vagrantup.com/ [accessed: 2017-02-14].

[6] "VirtualBox," Oracle, URL: https://www.virtualbox.org/ [accessed: 2017-02-14].

[7] C. Jämthagen, M. Hell, and B. Smeets, "A Technique for Remote Detection of Certain Virtual Machine Monitors," in Proceedings of the Third International Conference on Trusted Systems, ser. INTRUST'11. Berlin, Heidelberg: Springer-Verlag, 2012, pp. 129–137. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-32298-3_9

[8] J. C. Matherly, "SHODAN the computer search engine," URL: http://www.shodanhq.com [accessed: 2017-01-30].

[9] D. Moore, C. Shannon, D. J. Brown, G. M. Voelker, and S. Savage, "Inferring Internet Denial-of-service Activity," ACM Trans. Comput. Syst., vol. 24, no. 2, May 2006, pp. 115–139. [Online]. Available: http://doi.acm.org/10.1145/1132026.1132027

[10] "Metasploit," Rapid7, URL: https://www.metasploit.com/ [accessed: 2017-01-30].

[11] M. Mueter, F. Freiling, T. Holz, and J. Matthews, "High Interaction Honeypot Analysis Tool," URL: https://sourceforge.net/projects/hihat/ [accessed: 2017-02-14].

[12] J. Zhuge, T. Holz, X. Han, C. Song, and W. Zou, "Collecting Autonomous Spreading Malware Using High-interaction Honeypots," in Proceedings of the 9th International Conference on Information and Communications Security, ser. ICICS'07. Berlin, Heidelberg: Springer-Verlag, 2007, pp. 438–451. [Online]. Available: http://dl.acm.org/citation.cfm?id=1785001.1785045

[13] "Know Your Enemy: Sebek, A kernel based data capture tool," The Honeynet Project, Last Modified: 17. November 2003, URL: http://old.honeynet.org/papers/sebek.pdf [accessed: 2017-02-14].

[14] M. Dornseif, T. Holz, and C. N. Klein, "NoSEBrEaK - attacking honeynets," in Proceedings from the Fifth Annual IEEE SMC Information Assurance Workshop, 2004., June 2004, pp. 123–129.

[15] M. Puttaroo, P. Komisarczuk, and R. C. de Amorim, "Challenges in Developing Capture-HPC Exclusion Lists," in Proceedings of the 7th International Conference on Security of Information and Networks, ser. SIN '14. New York, NY, USA: ACM, 2014, pp. 334:334–334:338. [Online]. Available: http://doi.acm.org/10.1145/2659651.2659717

[16] C. Seifert, "Capture-bat download page," URL: https://www.honeynet.org/node/315 [accessed: 2017-02-14].

[17] A. Moshchuk, T. Bragin, S. D. Gribble, and H. M. Levy, "A Crawler-based Study of Spyware in the Web," in Proceedings of the Network and Distributed System Security Symposium, NDSS 2006, San Diego, California, USA. The Internet Society, 2006.

[18] J. R. Rocaspana, G. Portokalidis, P. Homburg, and H. Bos, "Shelia: a client-side honeypot for attack detection," 2009, URL: http://www.cs.vu.nl/˜herbertb/misc/shelia/ [accessed: 2017-02-14].

[19] J. Butler, "Bypassing 3rd Party Windows Buffer Overflow Protection," URL: http://phrack.org/issues/62/5.html [accessed: 2017-02-14].

[20] S. Jalayeri and T. Jarmuzek, "PwnyPot, High Interaction Client Honeypot," URL: https://github.com/shjalayeri/pwnypot [accessed: 2017-02-14].

 ISBN: 978-1-61208-564-7

Security Testing Over Encrypted Channels on the ARM Platform

Fatih Kilic
Chair for IT-Security
Technical University of Munich
Garching near Munich, Germany
e-mail: kilic@sec.in.tum.de

Benedikt Geßele
Department Secure Operating Systems
Fraunhofer AISEC
Garching near Munich, Germany
e-mail: benedikt.gessele@aisec.fraunhofer.de

Hasan Ibne Akram
Safety & Security Lab
Matrickz GmbH
Unterscheissheim near Munich, Germany
e-mail: hasan.akram@matrickz.de

***Abstract*—Security Testing has been applied for many years to detect vulnerabilities in applications. With the increasing demand for encryption to protect the confidentiality of network data, the requirements have changed. When proprietary, closed source software uses end-to-end encryption, security testing tools which are fuzzing the application layer over network with plaintext data will eventually fail. The Intrusion Detection Framework for Encrypted Network Data (iDeFEND) framework circumvents this problem without violating the security of the end-to-end encryption. Unfortunately, the framework cannot be used on the Advanced RISC Machines (ARM) platform, since it uses architecture depended features of x86. In this paper, we transfer iDeFEND to the ARM architecture and thereby, make it suitable for testing applications on embedded devices. In addition, we discuss the limitations of the current framework and improve it with novel methods to provide a more generic approach for security testing. We present a generic method for inspecting data on encrypted channels. Our approach does not require any knowledge of the structure of the wrapper function for receiving and decrypting like iDeFEND. Furthermore, we present a solution to test and inspect applications that are using packet queues. Finally, we evaluate our approach on popular mobile applications.**

Keywords–security testing; network security; reverse engineering; encrypted communication; embedded security.

I. INTRODUCTION

Nowadays, a wide variety of applications use encryption to protect their confidential data in network communications. Encrypting the network traffic prevents attackers from accessing sensitive data, but cannot stop them from exploiting security flaws in the implementation to achieve crashes, intrusion or code execution on the system. Security testing is responsible for detecting these vulnerabilities at an early stage. However, even powerful testing frameworks are blind when end-to-end encryption is applied and can only randomly generate or mutate packets. Additionally, the encryption layer makes it difficult for security testers to validate the remote program which increases the risk of missing faults. Solutions to this issue usually require a high amount of reverse engineering, since most of the target applications are closed source. Other solutions add an additional node to the encryption (e.g., a proxy server) and use it to access the plaintext data. This makes the communication more insecure. End-to-end encryption is designed to only terminate at the destination application to fulfil its required security. As a consequence, the plaintext can only be accessed by reverse engineering of the encryption algorithm and key, which is in general highly complex and time-consuming and thus, not feasible.

Another solution is presented by the generic framework iDeFEND [1]. The framework sustains the end-to-end encryption and leaves the communication channel untouched by extracting the plaintext data directly from process memory. It automates the reverse engineering process of applications by only relying on the detection and hooking of network and encryption functions. As a result, even closed source software can be handled at a much smaller effort. Although the framework has a generic design, it still has limitations. iDeFEND was implemented and evaluated for the x86 architecture, but nowadays most of the networking applications are running on mobile devices like smart phones or tablets whose processors are primarily designed by ARM. Since the framework uses hardware dependant features, its concept must be adapted to the specifics of the new platform.

Additionally, mobile applications tend to buffer network packets in a queue before sending them. This compensates bad connectivity, but results in a conflict with the current approach of iDeFEND. Furthermore, the framework relies on the presence of a specific wrapper function to inspect the received, unencrypted network data. In practice, this function can be more complex than expected by the framework and requires additional reverse engineering.

We overcome these shortcomings and extend the iDeFEND system. We provide a framework that allows to use common security testing tools for encrypted network applications. In summary, our contributions are the following.

- **Security testing over encrypted channels on ARM**
  We provide the same features of iDeFEND for ARM as it already does for x86. This means, we enable security testing on ARM devices when the target applications are communicating over an encrypted channel.
- **Improving iDeFEND to support applications with packet queues**
  We improve the current approach of iDeFEND with a new feature that makes it capable of handling applications with packet queues. Our new method allows to inject plaintext data into the packet queue and thus, into the encrypted communication channel.
- **Improving iDeFEND with a generic method for data inspection**
  We extend the concept of iDeFEND by a generic method for extracting received network data. We describe how this method enables the inspection of server responses without reverse engineering the function in detail.

The remainder of this paper is structured as follows. First, we present related work in Section II. In Section III, we summarize and describe the approach of the existing iDeFEND

 ISBN: 978-1-61208-564-7

framework. How the framework is used for security testing is explained in Section IV. In Section V, we present the limitations of the current concept and introduce our design improvements. In Section VI, we discuss the conceptual transfer of iDeFEND from x86 to ARM. The implementation of iDeFEND on ARM follows in Section VII. In Section VIII, we evaluate our framework and summarize the paper in Section IX.

II. Related Work

Many different fuzzing frameworks exist that facilitate the security testing of network communicating applications. Gascon et al. [2] present a fuzzing framework for proprietary network protocols which uses inference to create a generative model for message formats. Their approach relies on unencrypted network traffic, similar to many other smart automated model-based [3][4][5] and grammar-based [6][7] fuzzing techniques. Nowadays, there is also a vast amount of powerful commercial fuzzing and vulnerability scanning frameworks like Defensics [8], Nessus [9], beSTORM [10], Peach Fuzzer [11], honggfuzz [12] and american fuzzy lop [13] on the market available. They provide very complex and sophisticated algorithms to cover many different areas of fuzzing and vulnerability testing, but overall also lack proper support of encrypted network communications.

Biyani et al. [14] address this issue and present a solution by extending the SPIKE fuzzing framework to support encrypted protocols. They add a SSL wrapper to the existing plaintext fuzzer which allows to communicate with the target test application over an encrypted tunnel. This way, the fuzzer can inject its plaintext test data into the encrypted channel and test the target application for vulnerabilities. This approach, however, is limited to Secure Sockets Layer (SSL) encryptions which only represent a small part of proprietary software products. Another drawback is that their implementation is customized and only applicable for the open source fuzzer SPIKE. Tsankov et al. [15] introduce a different solution that allows a more generic fuzzing of encrypted protocols. Their approach is based on the knowledge of the encryption key and algorithm which is problematic from a security point of view.

As of yet, there is no good solution to testing of applications with encrypted network traffic. Our approach is different. We use the iDeFEND framework [1] to have a layer between test program and test framework. This additional layer makes the encryption transparent without violating the security of end-to-end encrypted communications. This way, we reduce the problem of testing encrypted protocols to the testing plaintext protocols and thus, enable the usage of many already existing testing tools.

III. Design of iDeFEND

In this section, we summarize the iDeFEND [1] framework and describe how the framework enables inspection and injection of plaintext data in encrypted communications. We also show why the approach is well suited for security testing.

Usually, applications implement encrypted communication with the help of two wrapper functions. One takes plaintext data, encrypts it and sends it over the network. This function is labelled EnCrypt & Send (CaS). The other one, Receive & DeCrypt (RaD), is responsible for the reversed process. It receives ciphertext data from the network and decrypts it afterwards. Together, these functions form the transition between plaintext and encrypted network data in our target applications. The iDeFEND framework uses this property to get access to the unencrypted network data by detecting and hooking both wrapper functions. This way, the application itself serves as an abstraction of the encryption implementation and allows us to inspect the plaintext communication without knowing the encryption algorithm, key or even source code of the application.

Controlling the wrapper functions empowers us to inspect, intercept, modify and inject new plaintext messages into the encrypted channel. For security testing, especially fuzzing, the tester primarily wants to send test data to the remote application and thus, heavily relies on the injection of packets. Since the CaS wrapper function takes a plaintext data pointer as argument, encrypts it and sends it over the network, test data can be injected by passing its pointer to the CaS. This can be realized in two different ways. Either active by code injection to the target process and calling CaS directly or passive by hooking calls to CaS inside the application (e.g., with a debugger) and replacing the input plaintext pointer with a pointer to the test data. In both scenarious, the test data is sent to the remote application, the response is extracted at the RaD and the test case can be evaluated.

The functionality of iDeFEND is logically split into three modules: a detector, a collector and a monitor module. The detector module is responsible for locating the wrapper functions in memory. Afterwards, the collector module hooks the located wrapper functions and passes the plaintext data to the monitor module. The monitor module simply is an interface for external programs. The detector module is a debugger that is specifically geared towards the automated reverse engineering of the wrapper functions. In general, applications with encrypted network traffic implement the functions *crypt*, *send* and *receive*. Send and receive are public library functions of the operating system and thus, getting their addresses is simple. The crypt function, depending on the underlying algorithm, can either be one or two functions. In case it is part of a library, getting the addresses is simple. They can be extracted by looking at the export table. In case it is not, the paper for interactive function identification [16] introduces an approach that facilitates the identification. By definition, the wrapper functions successively call the pairs encrypt and send, and receive and decrypt, respectively. iDeFEND uses this property of CaS and RaD to identify the wrapper functions through backtracking with a debugger. The backtracking is realized with breakpoints on send, receive, enrypt and decrypt. When the debugger notices a break on one of the function pairs, it can determine the wrapper functions from the call stack. Sometimes data is only encrypted for internal purposes and never sent over the network. In order to filter those cases, iDeFEND compares the data pointers between the function calls and validates the data flow. Data for network communication is detected, for instance, if the output pointer of the encryption matches the input pointer of the send. Otherwise, the calls of encrypt and send were independent and did not originate from the wrapper function, but from an internal encryption.

The collector module hooks the detected wrapper functions and extracts the network data. It is either part of the debugger or a module that is injected into the target application.

- **Collector Module as a Debugger**

Extracting the plain text with a debugger is simply achieved by inserting breakpoints on the wrapper functions CaS and RaD and extracting the data from their function arguments and return values, respectively. Since the debugging procedure is comparably slow, the target application is slowed down to a certain degree.

- **Collector Module as an Injected Module**
  A faster solution is to directly place code in the target application with a module injection. An assembly hook that is placed at the function prologue of the wrapper functions CaS and RaD redirects the execution to the injected code. The hook consists of a machine instruction like a jump or a call that substitutes the first few bytes of the function prologue and a function stub that is executed by the jump. The extracted plain text data then is passed via Inter Process Communication (IPC) to the monitor module.

IV. Security Testing with iDeFEND

In this section, we present an use case of the iDeFEND framework and explain how it enables security testing of encrypted network applications.

The iDeFEND framework is designed to support security testing of proprietary, closed source software. This type of testing is referred to as black box testing, since we examine the functionality of the programs under test without knowing details on the development, program internals or implementation. Even though the program is a blackbox, security analysts still can use powerful fuzzing tools to test for commonly known vulnerabilities. They can, for example, test a server against blind format string attacks [17]. In this scenario, a security analyst sends strings to the server application and afterwards validates the response and thereby, the outcome of the test case. For applications that use an encrypted communication channel, this approach of security testing inevitably fails. Since no information about implementation and design of the target application are available, also the internals of the encryption are unknown. As a result, there are only two possible responses of the target application to plaintext test messages from the security analyst. Either the test message does not fulfil the specification of the protocol and thus, the decryption fails and the test data is rejected. Or the decryption handles the test data, but changes it arbitrarily and is interpreted differently to the intentions of the tester. In both cases testing fails. Figure 1 illustrates this scenario with the orange arrow representing the test string data. The diverging arrow heads symbolize the misinterpreted test data after decryption that does not trigger the intended functionality any more.

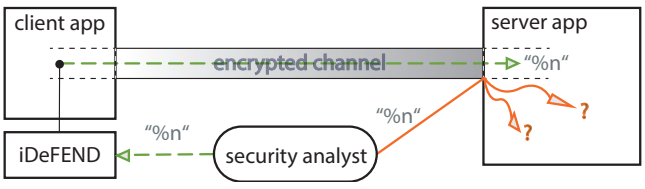


Figure 1. Security testing of encrypted communications.

If the security analyst wants to test the server application as intended, he can use the iDeFEND framework. Using the framework for testing circumvents the issue of encryption. It provides an interface for the security analyst to the client application and thus, access to the encrypted channel. This way, the security analyst can pass the plaintext test data to the framework interface which uses the client application to encrypt and send the data. The sent data then is decrypted correctly at the server application and eventually triggers the intended functionality. Figure 1 shows the flow of the plaintext test data with the dashed, green arrow. The security analyst passes the data to iDeFEND which injects it into the encrypted channel. The test data enters the server application and is decrypted correctly.

V. Improvements of iDeFEND

In this section, we discuss the limitations of the current iDeFEND approach for software testing and present our improvements. We put focus on the conceptual weaknesses of the framework and separately address the transfer to ARM in the following section VI.

Currently, iDeFEND implements the identification of the wrapper functions with backtracking. Therefore, the call stacks at successive calls to the logic function pairs are intersected. Knowing, for example, that wrapper CaS is responsible for calling encrypt and send, means that the call stacks of encrypt and send must have an intersection at the wrapper function. This approach introduces a weakness. The wrapper functions can only be detected when they successively call encrypt and send. For applications that use a message queue in network communication, this assumption is never met.

Additionally, iDeFEND defined the RaD wrapper function to return the decrypted plaintext packet. It inspects the plaintext data by hooking the function at its return instruction. This requires detailed knowledge about the structure of RaD and obviously the presence of a RaD.

In the following subsections we propose solutions to those two problems.

A. Test Data Injection into Packet Queues

Applications that use a packet queue construct the packet, encrypt it and then append it to the queue. At any other point in the program the encrypted packet is taken from the queue and sent over the network. As a result, the call graphs of encrypt and send do not intersect at the CaS, because there is no CaS any more. This introduces a weakness of the iDeFEND framework. Without the detection or presence of the wrapper function, the framework cannot inspect, intercept or inject data into the communication. This means, for applications that use packet queues it is not possible to use iDeFEND for security testing. We addressed this issue and analysed the program structure of such applications and came up with a solution. Even though the applications do not implement a CaS function, they still have a function that takes the plaintext data, encrypts it and appends it to the queue. This function can be used in the same manner as the CaS to inspect, intercept and inject data to the communication. The only difference is that the sending is delayed in time, which is irrelevant to our scenario of testing. Figure 2 illustrates the control flow graph for this new function type EnCrypt & EnQueue (CaQ). Identifying the address of this wrapper function requires a new approach. Usually, programs implement protocols that construct different packets for many different purposes. This means that for each packet the wrapper function is called from a different calling

 ISBN: 978-1-61208-564-7

context, but their call stacks always intersect at the CaQ. For this reason, our solution to the issue of identifying the CaQ function is to record all call stacks at encrypt and intersect them to find the wrapper function. In order to validate network traffic in this scenario, it is also necessary to use a different procedure to the previous. Since the data is copied to the queue, the pointers at send and encrypt vary. We handle this problem by not saving the pointer itself, but the whole buffer. At the validation of the data flow we simply compare the contents.

The CaQ function can be identified as soon as at least two call stacks from different calling contexts are collected. The intersection of the collected call stacks identifies the wrapper.

This proposed method extends iDeFEND to support applications that implement packet queues.

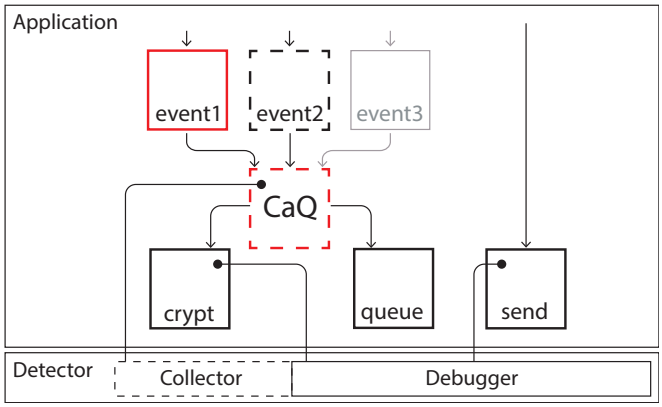


Figure 2. Control Flow Graph (CFG) for wrapper function CaQ.

B. Generic Approach for Data Inspection

The second problem of iDeFEND is that the current approach assumes the existence of a specially structured RaD function which in general is not the case. The RaD is assumed to return the decrypted plaintext data. iDeFEND hooks the RaD at the return and extracts the plaintext data. However, many applications do not implement this type of wrapper function. In general, the receiving wrapper function is a loop that never returns. It calls receive and passes the data to a parsing unit that finally decrypts the data. Furthermore, without knowing the structure of the RaD, the current iDeFEND cannot inspect the plaintext data. The correct offset and the information about the correct register or data pointer have to be known at this point.

We analysed this issue and came up with generic solution. Our approach is based on the assumption that data that is received over network is always decrypted at any later point in the program. Therefore, we store all incoming data at the function receive and wait for it to be decrypted. When the decryption function is accessing the data, we can extract the plaintext after the decryption has completed. This way, we do not need the presence of wrapper functions or knowledge about the function structure, but only require the presence of the the basic functions decrypt and receive. Additionally, our improved approach does not even rely on frame pointers.

Similar to the original approach, we also break on receive and decrypt. However, we identify data that is received from the network not by comparing the pointers of data, but by comparing the content of input and output buffer between receive and decrypt. The idea is the same as it was for validating data that is going to be send over the network for the CaQ. When the decrypt function returns and we validated that the encrypted data was received from the network previously, we extract the plaintext data from the returned buffer. The extracted data then can be passed to the tester for inspection.

With this method we extended iDeFEND to allow the inspection of unencrypted server responses, even though the application does not implement a wrapper function and use frame pointers.

VI. Transfer to the ARM Architecture

In this section, we discuss the transfer of iDeFEND to the ARM platform. We present the key differences between x86 and ARM with respect to debugging with hardware breakpoints, data extraction at function calls, call stack reconstruction from the stack and hooking of functions on machine code level.

A. Using Hardware Breakpoints for Debugging

Both architectures x86 and ARM implement both hardware and software breakpoints. Hardware breakpoints offer a better performance, do not require modification of the executable code and thus, are less obvious to detect. This makes them perfectly suited for implementing the detector module of iDeFEND.

In general, only a few hardware breakpoints are available per processor, but this is no limitation, since the specification of x86 offers up to four and ARM up to 16 hardware breakpoints. Implementing the detector requires at most four breakpoints. On x86, each debug register represents a breakpoint and holds the target address. A shared control register holds flags to enable, disable and configure each breakpoint. On ARM, hardware breakpoints consist of two registers: a control and a value register [18]. The value register stores the address of the breakpoint and the control register contains breakpoint options that allow, for example, to link different breakpoints, enable or disable them, specify the privilege and exception level the breakpoint debug event is generated on.

B. Extracting Data from Procedure Calls

The collector has to extract data from the function parameters on breaks. Since we break on function prologues, which means on the first instruction of the routine, we can access the passed parameters as specified by the underlying calling convention.

ARM, in contrast to x86, specified its own procedure call standard [19]. On ARM, the first four parameters are always passed in the first four registers `R0` to `R3`. Every additional parameter is pushed to the stack. Since the *Stack Pointer* register always holds the address of the top of the stack, the arguments five and higher can be accessed with help of the stack pointer register and the argument offset.

In case the output buffer is passed through the return value of a function, the *Link Register* can be used to access it. The *Link Register* is dedicated to hold the return address of the current function. The return value is passed through register *R0*.

 ISBN: 978-1-61208-564-7

TABLE I. PRESENCE OF STACK POINTERS WITH DIFFERENT COMPILER SETTINGS

GCC Flag	Optimization				Offset to next frame pointer (FP)
	O0	O1	O2	O3	
no flags	✓				FP - 4
-mapcs-frame	✓	✓	✓	✓	FP - 12
-fno-omit-frame-pointer	✓	✓	✓	✓	FP - 4
-mapcs-frame fno-omit-frame-pointer	✓	✓	✓	✓	FP - 12

C. Call Stack Reconstruction

In order to identify the wrapper functions CaS and RaD, we want to intersect the call stacks and therefore, have to reconstruct them from the program stack. In a program every function call pushes a frame to the stack and pops it on return. The call stack can be reconstructed by unwinding the stack frame by frame. On ARM, unwinding the stack is complex. In general, the architecture does not provide a dedicated frame pointer register for the address of the first frame. Depending on the optimization level of the underlying compiler, frame pointers might not even be present on the stack. This is problematic, since it is highly complex to reconstruct stack frames without having frame pointers, as it requires a sophisticated analysis of the stack. Table I illustrates the effects of different settings on the generation of stack frames for the GCC compiler. The flags *mapcs-frame* and *fno-omit-frame-pointer* force the compiler to preserve stack pointers throughout all optimization levels. The only difference is that the pointer offsets vary. Without them, the compiler only generates stack pointers for optimization level O0, which means no optimization. In the default case, without any particular flag specified, frames are properly build by the compiler.

D. Hooking Functions

Injecting the collector module into the target process requires a redirection of the control flow from the original code to the injected module. Therefore, a hook is placed in the executable code at the prologue of the target function. Generally speaking, this means substituting the first bytes of the function prologue with a branch instruction. The replaced code must be backed up and executed later on, before jumping back to the original function.

On ARM, instructions have a fixed length of four bytes, which makes substitution of instructions simple. However, multiple types of prologues exist. This is problematic when the first instruction is program counter dependant and thus, cannot be moved. This happens on ARM, for example, when compilers use constant pools. Otherwise, when the instruction is independent of the program counter, the instruction can be moved and a hook is possible.

The actual branch can be implemented with a memory load that allows to target the full 32 bit address space. Since it modifies the program counter directly, the hook consists of only one instruction plus memory space that is holding the target address. Since compilers use multiple bytes of padding between two procedures in memory, this padding is a suitable location to place the address.

VII. IMPLEMENTATION

We implemented the improved iDeFEND framework on an ARM device that is running a Linux operating system. We chose Linux, as most of the target ARM devices like smart phones, tablets or embedded boards are either running Linux or Android, which is also based on the Linux Kernel. We used a Raspberry Pi 2 embedded board that is equipped with a 900MHz quad core ARM Cortex-A7 processor and 1GB RAM. It was running a Linux distribution Raspbian 4.1.13-v7 as operating system. For the sake of efficiency, portability to Windows and independence of other programs and their implementations, we decided to write our prototype as a stand alone C program.

The implementation consists of two parts. First we present the detector module, followed by the implementation of the collector module.

A. The Detector Module - a Debugger based on ptrace

We implemented the detector on top of the *ptrace* debugging API by setting four hardware breakpoints for each of the target functions.

1) Finding addresses of Send and Receive: Since Linux maps the whole binary object to memory, the virtual addresses of send and receive can be calculated by adding the offsets in the binary to the base address of the module process space. We retrieve the base address and path to the binary on disk from the directory */proc*. We then use the utility *nm* to find the offsets inside the binary.

2) Detecting Successive Calls to Function Pairs and Locating the Wrapper Functions: In order to locate the wrapper functions, we identify successive calls by extracting the function arguments at encrypt and at receive, and see if they match the input pointers at send and decrypt. For the special CaQ case, we copy the whole buffer instead of only pointers. We track the data per thread, together with a time stamp. A 15 seconds time out prevents internal encryptions to stay infinitely long in memory. We implemented stack unwinding for applications compiled with *-mapcs-frame*. As described in Table I, each frame pointer minus twelve then points to the previous pointer. After reconstruction, we intersect two call stacks by searching for the first frame that appears in both call stacks.

B. The Collector Module - Speed Up with Module Injection

We implemented the collector in both variants debugger and injected module. For injection, we implemented a call to *dlopen* that uses the dynamic loader of Linux to load objects at runtime. Finally, we placed the hooks at the wrapper functions and detoured the execution to the injected module.

VIII. EVALUATION

We have evaluated our improved iDeFEND framework for five applications. Beside the required criterion of encrypted network communication, we wanted to have at least one messenger, one file transfer and one secure shell application. These types implement different network protocols which handle text messages, binary files and customized commands. Furthermore, we wanted to have at least one test application that is single-threaded, multi-threaded, uses the console for user interaction and implements an own Graphical User Interface (GUI). In order to have ground-truth information of the wrapper functions, we used open source applications. Table II gives an overview of the selected applications telegram-cli (v1.4.1), uTox (v0.7.1), PLINK (v0.67), PSFTP (v0.67)

 ISBN: 978-1-61208-564-7

TABLE II. DESCRIPTION OF THE OPEN SOURCE TEST APPLICATIONS THAT RUN ON A RASPBERRY PI 2

Name	Type	Data Category	UI	Threading
telegram-cli	Messenger	Text	Console	Multi
uTox	Messenger	Text	GUI	Multi
PLINK	Secure Shell	Commands	Console	Single
PSFTP	File Transfer	Files	Console	Single
PSCP	File Transfer	Files	Console	Single

TABLE III. EVALUATED APPLICATIONS

Name	Send	Receive	Wrapper Type
telegram-cli	Write	Read	CaQ
uTox	SendTo	RecvFrom	CaS
PLINK	Send	Recv	CaS
PSFTP	Send	Recv	CaS
PSCP	Send	Recv	CaS

and PSCP (v0.67). The second column states the type of the application. The third column shows the type of data that is primarily transferred by the protocol. The last two columns indicate whether the application implements a GUI or is multi or single threaded, respectively.

Table III summarizes the results of our evaluation. The first column contains the names of the applications. The columns send and receive state the system library functions the application used to communicate over the network. The column Wrapper-Type states whether a CaS or CaQ function is implemented. Briefly summarized, we were able to inspect, intercept and inject data for all five applications. Except for Telegram, all applications implement the CaS function. Telegram implements a message queue and therefore, uses the CaQ. With the improved approach we were able to detect it and use it for packet injection. We were also able to use code injection and hooking of the wrapper functions on all five applications.

IX. CONCLUSION

With the rising demand for confidentiality and thus, encryption in consumer level and commercial software, security testing faces new challenges. Currently, existing testing tools only have poor or no support at all for encrypted network communications. That is precisely the reason why we proposed a generic solution to this issue by using the iDeFEND framework. The framework makes the encryption transparent and thereby, does not violate the security of end-to-end encryption. Since iDeFEND cannot be used on the ARM platform and nowadays many network applications are from the mobile sector and thus, use ARM processors, we transferred it to the this architecture. Additionally, we pointed out the limitations of the current framework and introduced improvements to it. Our novel methods provide a more generic approach for security testing. We introduced a method that allows to inject test data into network applications that use message queues. Our solution detects and hooks the function that is responsible for encrypting and enqueuing packets.

Furthermore, we introduced a generic method to inspect the incoming unencrypted network data. Our method does not rely on the presence of a receive and decrypt wrapper function or even frame pointers.

With the extended iDeFEND framework we provide an interface to the encrypted channel of an application that allows already existing testing tools to work as intended, also on the ARM platform. Our improved framework decouples the testing of software from the actual encryption.

REFERENCES

[1] F. Kilic and C. Eckert, “idefend: Intrusion detection framework for encrypted network data,” in Proceedings of the 14th International Conference on Cryptology and Network Security (CANS 2015), ser. Lecture Notes in Computer Science. Springer International Publishing, 2015, vol. 9476, pp. 111–118.

[2] H. Gascon, C. Wressnegger, F. Yamaguchi, D. Arp, and K. Rieck, Pulsar: Stateful Black-Box Fuzzing of Proprietary Network Protocols. Cham: Springer International Publishing, 2015, pp. 330–347.

[3] T. Kitagawa, M. Hanaoka, and K. Kono, “Aspfuzz: A state-aware protocol fuzzer based on application-layer protocols,” in Computers and Communications (ISCC), 2010 IEEE Symposium on, June 2010, pp. 202–208.

[4] G. Banks et al., SNOOZE: Toward a Stateful NetwOrk prOtocol fuzZEr. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 343–358.

[5] S. Gorbunov and A. Rosenbloom, “Autofuzz: Automated network protocol fuzzing framework,” IJCSNS, vol. 10, no. 8, 2010, p. 239.

[6] D. Yang, Y. Zhang, and Q. Liu, “Blendfuzz: A model-based framework for fuzz testing programs with grammatical inputs,” in 2012 IEEE 11th International Conference on Trust, Security and Privacy in Computing and Communications, June 2012, pp. 1070–1076.

[7] P. Godefroid, A. Kiezun, and M. Y. Levin, “Grammar-based whitebox fuzzing,” in Proceedings of the 29th ACM SIGPLAN Conference on Programming Language Design and Implementation, ser. PLDI ’08. New York, NY, USA: ACM, 2008, pp. 206–215.

[8] Codenomicon. Defensics. [Online]. Available: www.codenomicon.com/defensics/ 2016.04.26

[9] T. N. Security. Nessus. [Online]. Available: www.tenable.com/de/nessus 2016.04.26

[10] B. Security. bestorm software security testing tool. [Online]. Available: http://www.beyondsecurity.com/bestorm.html 2016.04.26

[11] Peach fuzzer. [Online]. Available: http://www.peachfuzzer.com/ 2016.04.26

[12] honggfuzz: A general-purpose, easy-to-use fuzzer with interesting analysis options. [Online]. Available: https://github.com/google/honggfuzz 2016.04.26

[13] M. Zalewski. American fuzzy lop: a security-oriented fuzzer. [Online]. Available: http://lcamtuf.coredump.cx/afl/ 2016.04.26

[14] A. Biyani et al., “Extension of spike for encrypted protocol fuzzing,” in 2011 Third International Conference on Multimedia Information Networking and Security, Nov 2011, pp. 343–347.

[15] P. Tsankov, M. T. Dashti, and D. Basin, “Secfuzz: Fuzz-testing security protocols,” in Automation of Software Test (AST), 2012 7th International Workshop on, June 2012, pp. 1–7.

[16] F. Kilic, H. Laner, and C. Eckert, “Interactive function identification decreasing the effort of reverse engineering,” in Proceedings of the 11th International Conference on Information Security and Cryptology (Inscrypt 2015). Springer International Publishing, 2016, pp. 468–487.

[17] F. Kilic, T. Kittel, and C. Eckert, “Blind format string attacks,” in Proceedings of the 10th International Conference on Security and Privacy in Communication Networks (SecureComm 2014), ser. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering. Springer International Publishing, 2015, vol. 153, pp. 301–314.

[18] ARM Architecture Reference Manual - ARMv8, for ARMv8-A architecture profile. ARM Limited, Jun. 2016.

[19] Procedure Call Standard for the ARM Architecture. ARM Limited, Nov. 2015, document Version: ARM IHI 0042F, current through ABI release 2.1.

 ISBN: 978-1-61208-564-7